Naval Command,
Control and Ocean
Surveillance Center RDT&E Division San Diego, CA
92152-5001

Technical Report 1682
October 1994

Development of an Artificial Neural Network for Real-time Classification of Cone Penetrometer Strain Gauge Data

John M. Andrews
Computer Sciences Corporation

Stephen H. Lieberman
NCCOSC RDT&E Division

Technical Report 1682
October 1994

Development of an Artificial Neural Network for Real-time Classification of Cone Penetrometer Strain Gauge Data

John M. Andrews
Computer Sciences Corporation

Stephen H. Lieberman
NCCOSC RDT&E Division

NAVAL COMMAND, CONTROL AND OCEAN SURVEILLANCE CENTER
RDT&E DIVISION
San Diego, California 92152–5001

K. E. EVANS, CAPT, USN
Commanding Officer

R. T. SHEARER
Executive Director

ADMINISTRATIVE INFORMATION

The work in this document was a combined effort of personnel from the Environmental Chemistry/Biotechnology Branch (Code 521) of the NCCOSC RDT&E Division (NRaD) and Computer Sciences Corporation. Sponsorship was provided by the Strategic Environmental Research and Development Program (SERDP), and the Naval Facilities Engineering Command (NAVFACENGCOM).

Released by
J. G. Grovhoug, Acting Head
Environmental Chemistry/
Biotechnology Branch

Under authority of
P. F. Seligman, Head
Environmental Sciences
Division

Technology detailed in this document is covered by a U.S. Patent Application assigned to the U.S. Government. Parties interested in licensing this technology may direct inquiries to:

Harvey Fendelman
Legal Counsel for Patents
Code 0012
NCCOSC, RDT&E Division
San Diego, CA 92152–5765
(619) 553–3001

PK

SUMMARY

OBJECTIVE

The purpose of this study was to develop a neural-network-based computer program for determining soil behavior type from cone penetration strain gauge measurements.

CONCLUSIONS

The neural-network-based algorithm for soil classification has been implemented and is currently meeting all performance expectations. However, Site Characterization and Analysis Penetrometer System (SCAPS) needs may change and it is impossible to anticipate all future requirements. At some later date, improvements or changes to the original program may be necessary.

RECOMMENDATIONS

Improving the accuracy up to 100% is certainly possible. The program may also be made smaller and faster. Each of these considerations may be investigated by performing a more thorough study of results obtained by using different network architectures and connectivity. Also, given the one-of-n encoding of the output variable, use of the SOFTMAX activation function in the output layer may help improve training.

If it becomes necessary to change the input units from tons per square foot (TSF) to some other measure of force, either the input to TSF must be rescaled or another network retrained. Making use of additional input parameters, such as pore pressure or a depth correction factor, will require complete retraining of the network.

CONTENTS

INTRODUCTION 1

BACKGROUND 1

SCAPS PROGRAM 1

NETWORK DEVELOPMENT 3

PREPARATION OF TRAINING DATA 3

NETWORK ARCHITECTURE AND TRAINING 3

RESULTS 5

CONCLUSIONS/RECOMMENDATIONS 10

ACRONYMS 11

REFERENCES 11

APPENDIX A: SOURCE CODE A-1

FIGURES

1. Simplified soil-behavior-type classification chart for standard electric friction cone 2

2. Data points (represented by +) causing *CPTINT* errors 4

3. Root-mean-square output error versus number of training iterations. Data shown are for a three-layer 2x9x12 neural network 5

4. Data points (represented by +) misclassified by a 2x9x12 single hidden-layer neural network after training 7

5. Data points (represented by +) misclassified by a 2x21x12 single hidden-layer neural network after training 7

6. Data points (represented by +) misclassified by a 2x11x9x12 two hidden-layer neural network after training 9

7. Data points (represented by +) misclassified by a 2x15x15x12 two hidden-layer neural network after training. Of all the networks trained, this one is the most accurate with a classification accuracy rate of 98.2% 9

TABLES

1. Neural networks with one hidden layer 6

2. Neural networks with two layers of hidden processing elements 8

INTRODUCTION

BACKGROUND

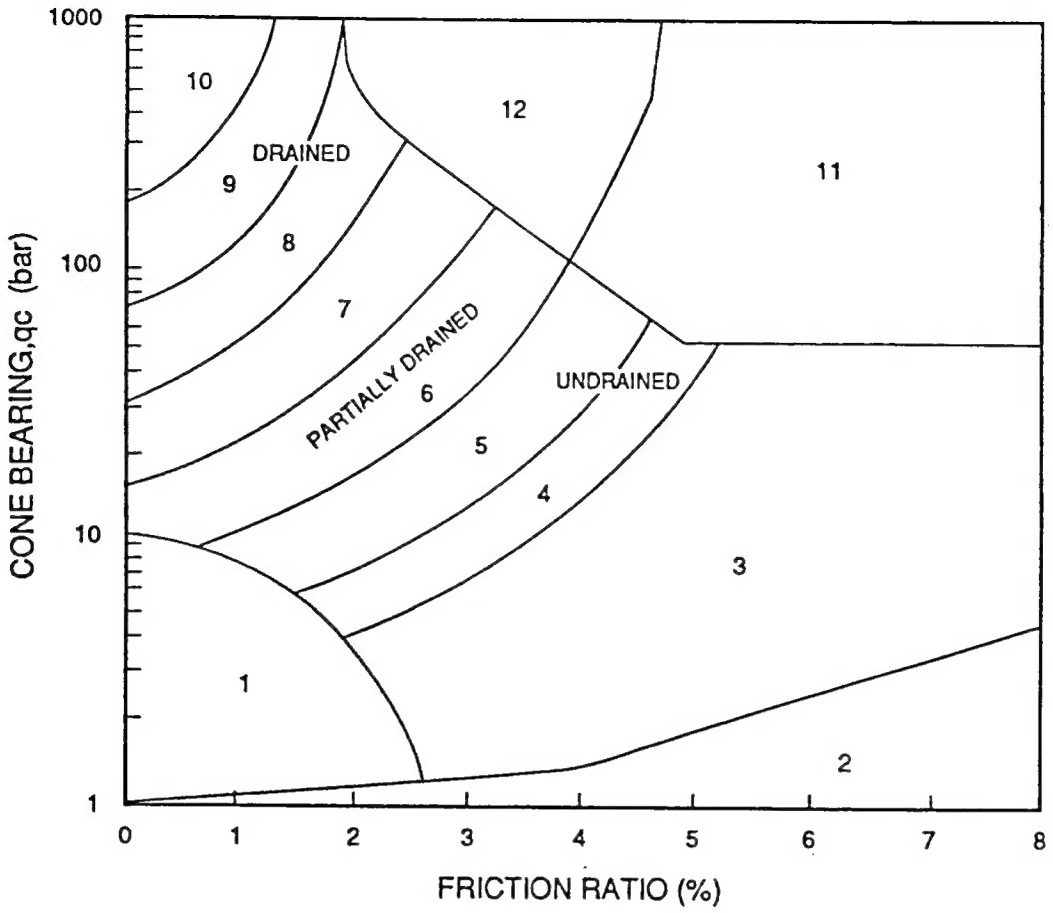
The purpose of this study was to develop a neural-network-based computer program for determining soil behavior type from cone penetration strain gauge measurements. The primary requirements were that the algorithm operate online in real time to provide highly accurate and verifiable determinations of 12 standard geotechnical soil classifications. Once developed, the program was to be integrated into the data acquisition software of the NRaD prototype Site Characterization and Analysis Penetrometer System (SCAPS). This manuscript, which serves to document both the network development process and the resulting C source code, may be used as a guideline for the future development of improved or modified versions of the original algorithm.

SCAPS PROGRAM

SCAPS is a modified truck-mounted cone penetrometer system that utilizes fiber-optic-based chemical sensors to measure underground contaminants *in situ* (reference 1). SCAPS maintains the ability to perform stratigraphic profiling by electric cone penetration testing while simultaneously gathering spectroscopic data. The electric cone penetration test (CPT) is achieved by measuring the response of two strain gauges mounted inside the penetrometer probe. As the probe is hydraulically pressed into the ground, strain gauges measure the pressure incident upon the cone-shaped probe tip and the frictional resistance along the probe axis. The CPT strain gauge measurements may be evaluated to determine soil classification type. Classification charts, such as the one shown in figure 1, can be used for this purpose (reference 2).

SCAPS was conceived of as a tool to provide an *immediate* onsite assessment of environmental contamination through the data acquired by its sensors. Hence, the need for a soil classification method that operates online in real time to convert raw CPT data to soil class information. No direct functional relationship exists for converting CPT values to soil behavior type. A look-up table could be programmed to perform this task; however, memory requirements for such a table would make this method difficult to incorporate into existing SCAPS data acquisition and control software.

It was decided to develop a software model of an artificial neural network to perform a mapping of CPT data to soil classification. One attractive aspect of this approach was that a neural network could learn the mapping algorithm by example. Expert knowledge in soil classification would not be necessary. Also, the network would be fast enough, even in its serial implementation, to run in real time. In addition, the memory requirements of the executable code are typically small enough to be easily added onto existing SCAPS software.



Zone	Soil Behavior Type
1	sensitive fine grained
2	organic material
3	clay
4	silty clay to clay
5	clayey silt to silty clay
6	sandy silt to clayey silt
7	silty sand to sandy silt
8	sand to silty sand
9	sand
10	gravelly sand to sand
11	very stiff fine grained (*)
12	sand to clayey sand (*)

(*) overconsolidated or cemented

Figure 1. Simplified soil-behavior-type classification chart for standard electric friction cone.

NETWORK DEVELOPMENT

PREPARATION OF TRAINING DATA

Neural networks require the use of training data to adaptively learn a relationship. Training data for neural networks of the type used with this project consist of a set of input data points paired with the appropriate class association (desired network output). In this instance, the input data includes the two strain gauge measurements, while the associated output corresponds to the derived soil classification. The training data used in this study were generated through the application program *CPTINT.* A CPT data interpretation program available from the University of British Columbia, *CPTINT* takes strain measurements as input parameters and reports the associated soil classification (reference 3). In that way, *CPTINT* is functionally similar to the neural algorithm being developed here. However, it is limited to offline data analysis using file input and output.

The CPNN program (appendix A) was written to generate artificial strain gauge readings and provide an input data file for *CPTINT.* CPNN creates 12,120 equidistant pairs of simulated strain gauge data points. The range for generated cone pressure values is 0 to 1000. Sleeve friction values range from 0 to 8% of cone pressure. The cone pressure values are unitless because *CPTINT* can be set to interpret data in terms of various pressure units. CPNN also associates a dummy depth variable with each data pair. This a requirement of *CPTINT.*

The output file generated by *CPTINT* consists of the original input values along with integer values ranging from 1 to 12, corresponding to the interpreted soil classification. Of the 12,120 input records, *CPTINT* generated an error response for 223; a 1.8% error rate. The errors were transmitted as soil class 1e99. As seen in figure 2, most of the errors occurred along the exterior boundary of the soil classification chart. These are presumably out-of-range errors. Other errors occurred along the decision boundaries between classification regions.

The file was edited for use as a formatted neural network training file using *DataSculptor,* a visual programming tool for data processing and analysis (reference 4).

DataSculptor was used to cull the error causing records from the training set. It was also used to dimension the original cone pressure values to log cone pressure in tons per square foot (TSF). The soil class encoding was changed from a 1 to 12 scalar value (thermometer code) to a 12-digit one-of-n coding. To avoid network saturation during training, the input parameters were linearly scaled to values between –0.8 and +0.8.

NETWORK ARCHITECTURE AND TRAINING

NeuralWorks Professional II/Plus[2] neural network development tool was used for network development. The backpropagation paradigm was selected. There is no known method for making *a priori* determinations of the optimal neural architecture. Hence, several networks designs based upon variations of the general feedforward backpropagation theme were trained and tested. Networks with one and two hidden layers of processing elements were investigated. Single hidden-layer networks where the input nodes are connected to both hidden and output layers *(NeuralWorks* connect prior option) were also tested. The initial weight values were also randomly varied since training performance is somewhat dependent upon starting conditions.

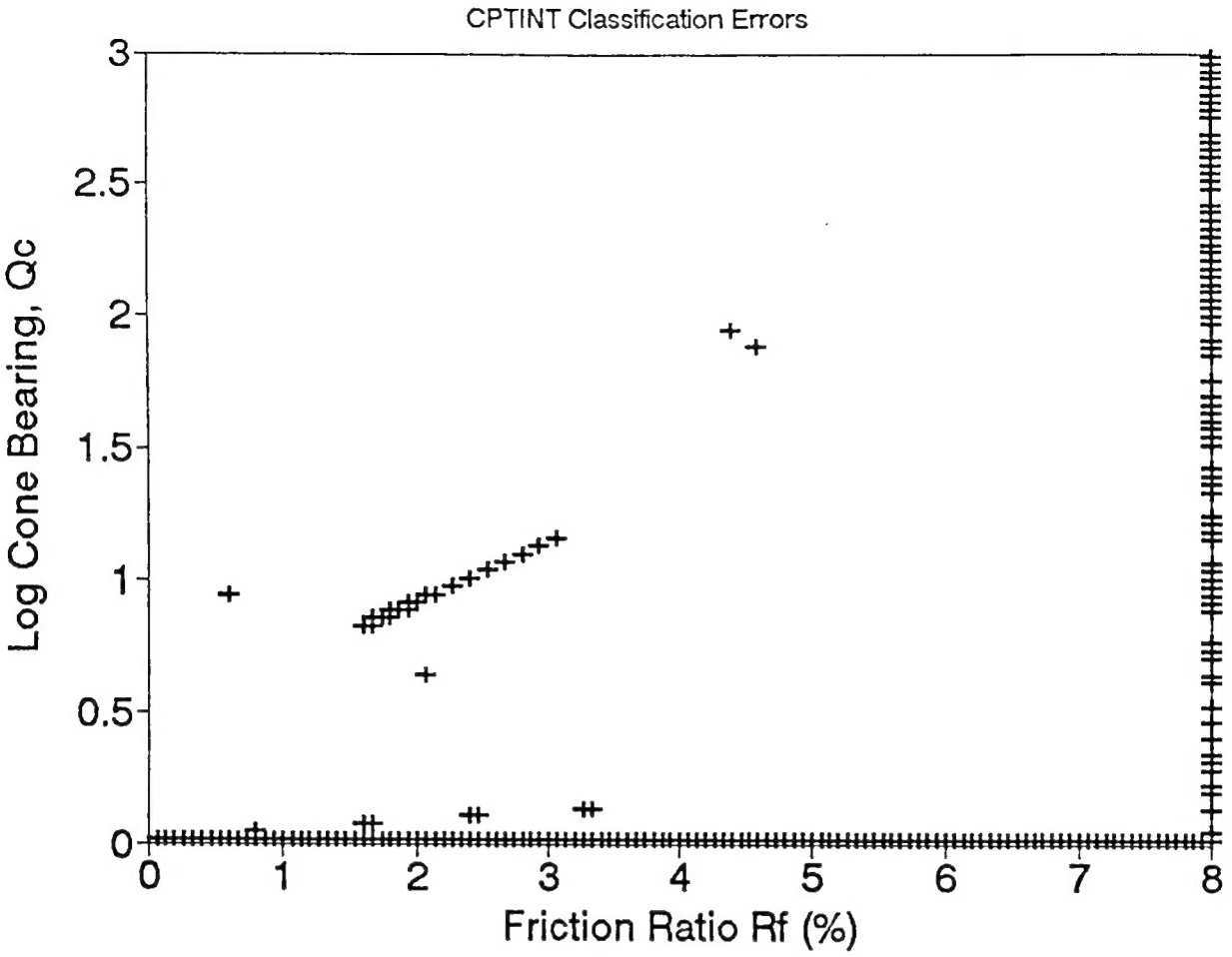


Figure 2. Data points (represented by +) causing *CPTINT* errors.

The network input layer for each of the networks consisted of two processing elements corresponding to the two input parameters; log cone pressure and friction ratio. The input units were set to TSF. The network output layer consisted of 12 processing elements corresponding to the 12 possible soil classes as represented by a one-of-n code. The output layer was competitive in that only the processing element with the highest activation output a 1. All others output a zero. The following training parameters were used:

learning coefficients at start:
first hidden layer: 0.4
second hidden layer: 0.35
output layer: 0.3
momentum coefficient at start: 0.4
transition points: 25000; 75000; 175000 iterations
learning rule: normalized cumulative delta (epoch = 16)
transfer function: Hyperbolic tangent

Each of the networks was trained for a total of 300,000 iterations. Additional training yielded no significant improvement in performance. Furthermore, since the set of all possible input values has a limited and well defined range, and since this set was uniformly represented by a very large set of training data, there was no possibility of "overtraining" the networks. Overtraining refers to a situation in which a network has essentially memorized the training data and is unable to make generalizations with new data.

RESULTS

Figure 3 follows the root-mean-square (RMS) training error as training proceeds for a three-layer (single hidden-layer) network. The shape of the curve signifies the rate at which the network learned, and at what point the weight values stabilized. It is typical of the other networks, both the three- and four-layer versions, in that most of the learning took place within the first 100,000 iterations.

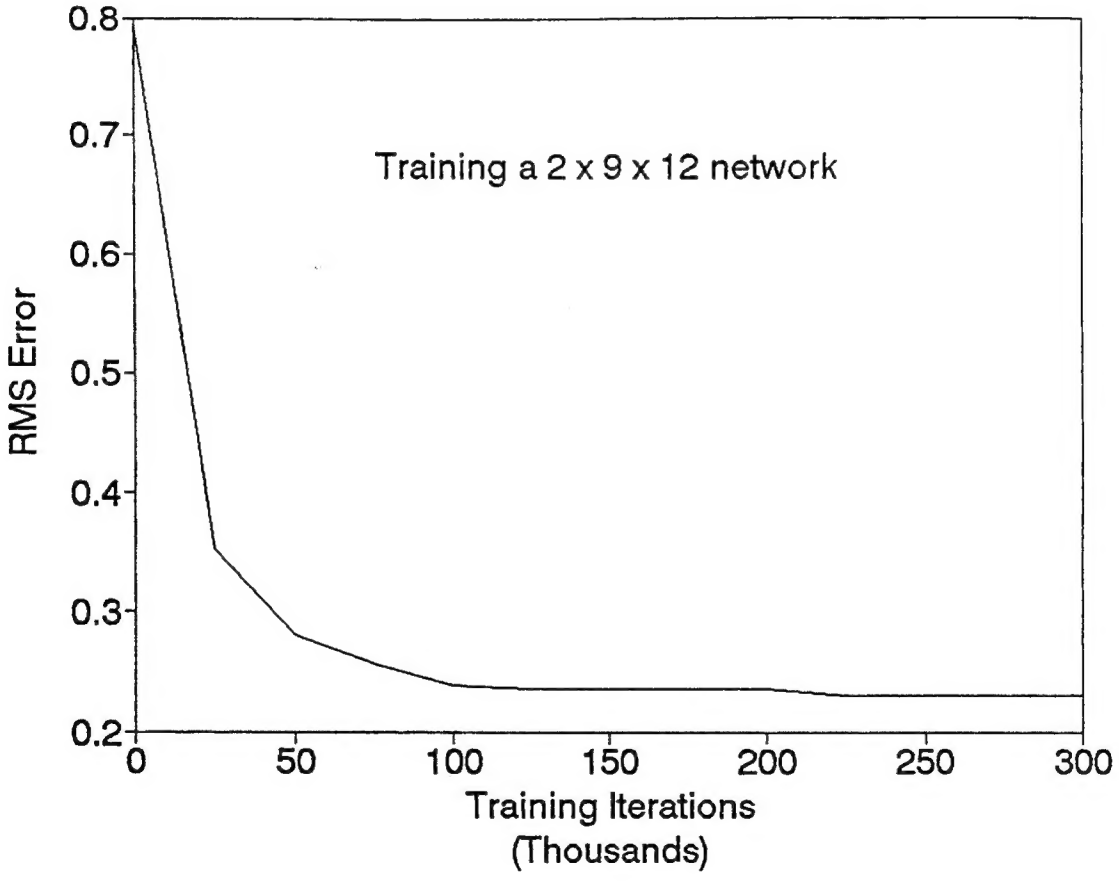


Figure 3. Root-mean-square output error versus number of training iterations. Data shown are for a three-layer 2x9x12 neural network.

After training, each network was tested on the same input data used for training. During the testing phase, actual network output is recorded to a result file. Program Start.C (appendix A) was written to sort through the *NeuralWorks'* result files and count the number of correctly and incorrectly classified data points. These numbers were then used to compare the performance differences of the various trained networks.

The RMS output errors for the single hidden-layer networks are shown in table 1. Networks with the same number of layers and processing elements differed only in the random weight values assigned to each connection prior to the start of training. It can be seen that the networks with direct connections between the input and output layers (connect prior) consistently performed worse than the other networks. The incorrectly classified data pairs for two of the networks are displayed in figures 4 and 5. Note that all of the errors occur near the decision boundaries. Given the overall number of errors and the breadth of the error regions, the classification performance of this series of networks is considered poor.

Table 1. Neural networks with one hidden layer. Columns correspond to the root name of associated files, the number of processing elements in the hidden layer, special architectural features, and the RMS error across the training net.

File Name	H. PEs	Special	RMS ERROR
CPT01	7		.23
CPT02	7		.24
CPT03	7		.23
CPT04	7		.24
CPT05	9		.22
CPT06	9		.23
CPT07	11		.21
CPT08	11		.22
CPT09	13		.23
CPT10	13		.22
CPT11	15		.23
CPT12	15		.22
CPT13	21		.22
CPT14	21		.22
CPT15	21	Connect Prior	.35
CPT16	21	Connect Prior	.35
CPT17	15	Connect Prior	.35
CPT18	15	Connect Prior	.35
CPT19	11	Connect Prior	.35
CPT20	11	Connect Prior	.35

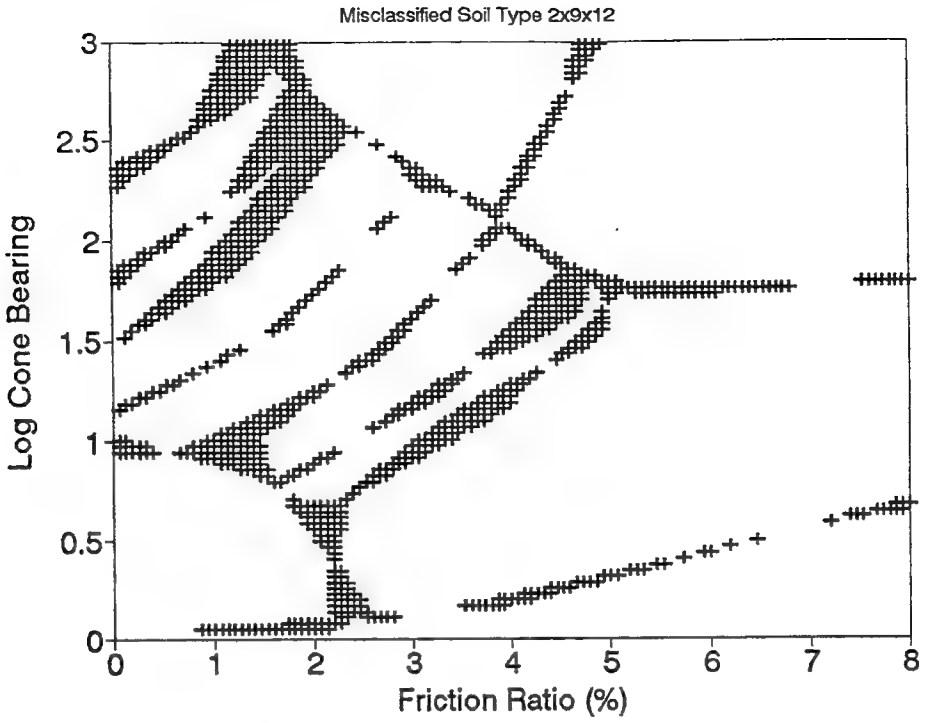


Figure 4. Data points (represented by +) misclassified by a 2x9x12 single hidden-layer neural network after training.

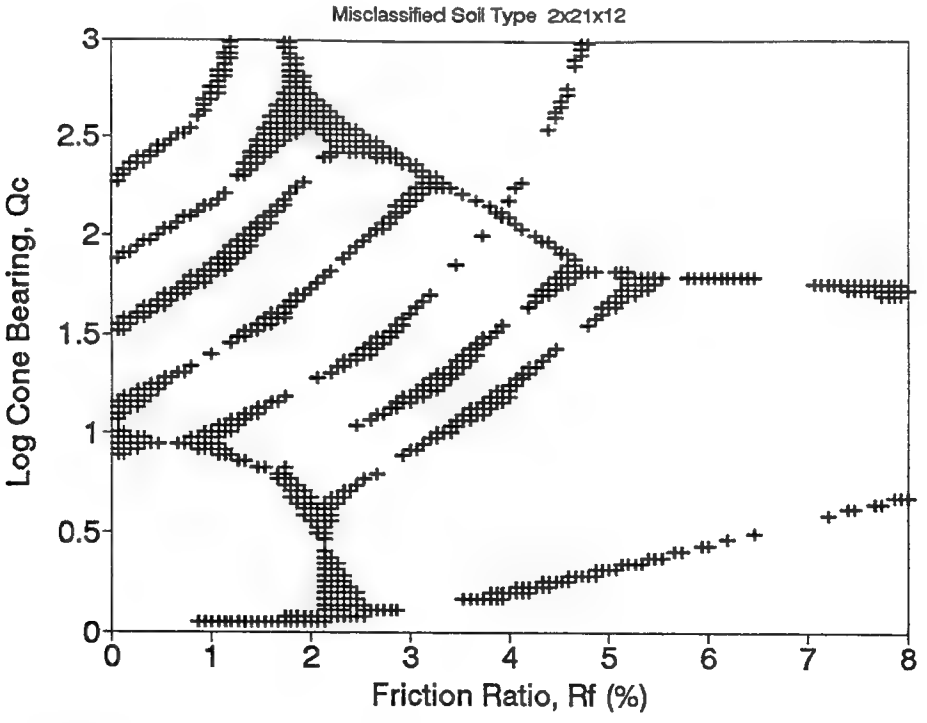


Figure 5. Data points (represented by +) misclassified by 2x21x12 single hidden-layer neural network after training.

The networks utilizing two hidden layers outperformed the single hidden-layer networks. Table 2 shows the classification results and accuracy for the networks with two hidden layers. The misclassified points for two of the networks are shown in figures 6 and 7. Again, the errors occur on the boundaries between class regions. For example, at the boundary between class 1 and 2, a class 1 soil may be classified as a class 2. This is a far more acceptable type of error than a class 1 soil being incorrectly identified as a nonneighboring class. This is particularly true because the soil class boundaries are considered somewhat inexact (fuzzy) anyway with some overlap between zones.

Table 2. Neural networks with two layers of hidden processing elements. Columns correspond to the root name of associated files, the number of processing elements in each hidden layer (layer two x layer three), the number of correct and incorrect soil classifications across the training set, and prediction accuracy after training.

File Name	H. PEs	Correct	Wrong	Accuracy
CP5x9a	5x9	10943	954	92.0%
CP5x9b	5x9	10155	1742	85.4%
CP7x7a	7x7	10923	974	91.8%
CP7x7b	7x7	11136	761	93.6%
CP9x5a	9x5	10282	1615	86.4%
CP9x5b	9x5	10311	1586	86.7%
CP9x7a	9x7	1026	871	92.7%
CP9x7b	9x7	11164	733	93.8%
CP10x8a	10x8	11377	520	95.6%
CP10x8b	10x8	11117	780	93.4%
CP11x7a	11x7	11087	810	93.2%
CP11x7b	11x7	11255	642	94.6%
CP11x9a	11x9	11587	310	97.4%
CP11x9b	11x9	11443	454	96.2%
CP11x11a	11x11	11559	338	97.2%
CP11x11b	11x11	11538	359	97.0%
CP15x15a	15x15	11579	318	97.3%
CP15x15b	15x15	11682	215	98.2%
CP21x21a	21x21	11656	241	98.0%
CP21x21b	21x21	11638	259	97.8%
CP50x50	50x50	11613	284	97.6%

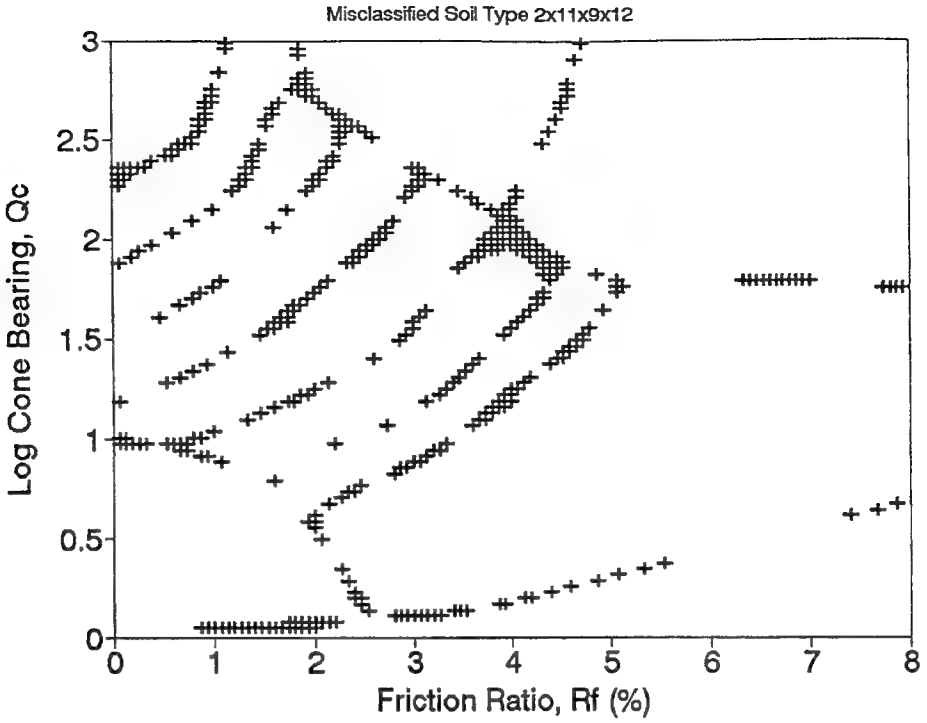


Figure 6. Data points (represented by +) misclassified by a 2x11x9x12 two hidden-layer neural network after training.

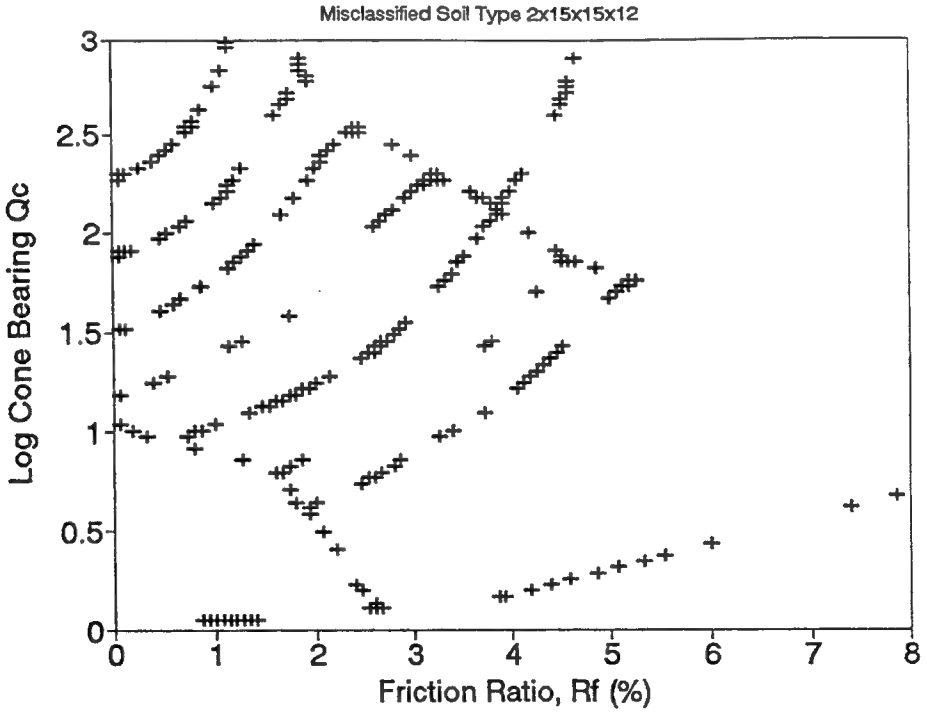


Figure 7. Data points (represented by +) misclassified by 2x15x15x12 two hidden-layer neural network after training. Of all the networks trained, this one is the most accurate with a classification accuracy rate of 98.2%.

The best performing network made use of a four-layer 2x15x15x12 architecture. This network responds with over 98% accuracy. That is, its classifications agree with the *CPTINT* program in 98.2% of the cases. Close-in performance was a 2x11x9x2 network with a 97.4% accuracy rate. The number of processing element interconnections for the smaller of these two networks is 263, counting the bias element. The larger network contains 479 interconnections. This computational overhead corresponds to nearly twice the memory requirements and half the recall speed. However, it was decided to incorporate the 2x15x15x12 network into the SCAPS software anyway. It is still sufficiently fast to run in real time and the memory requirements are reasonable. There is no need to compromise and give up the better classification performance.

The developed network was output in C code via *Neuralware* Flashcode option. Some modifications were made to clean up the code and make it more readable. The resulting program, SOIL.C, is reproduced in appendix A. The program was ported to BASIC by Palladin Software (San Diego, CA) for compatibility with other SCAPS software. The BASIC version was incorporated into SCAPS software in October 1993. Preliminary reports from initial SCAPS field studies at Naval Air Station North Island show excellent agreement between the neural-network-based CPT soil classification and laboratory classification of sample soils.

CONCLUSIONS/RECOMMENDATIONS

The neural-network-based algorithm for soil classification has been implemented and is currently meeting all performance expectations. However, SCAPS needs may change and it is impossible to anticipate all future requirements. It may be necessary to make improvements or changes to the original program at some later date.

Improving the accuracy up to 100% is certainly possible. The program may also be made smaller and faster. Each of these considerations may be investigated by performing a more thorough study of results obtained by using different network architectures and connectivity. Also, given the one-of-n encoding of the output variable, use of the SOFTMAX activation function in the output layer may help improve training.

If changing the input units from TSF to some other measure of force becomes necessary, either the input to TSF must be rescaled, or another network retrained. Making use of additional input parameters, such as pore pressure or a depth correction factor, will require complete retraining of the network.

ACRONYMS

CPT	Cone Penetration Test
RMS	Root-Mean-Square
SCAPS	Site Characterization and Analysis Penetrometer System
TSF	Tons per Square Foot

REFERENCES

1. Lieberman, S. H., S. E. Apitz, L. M. Borbridge, and G. A. Theriault, International Conference on Monitoring Toxic Chemical and Biomarkers, *EOS/SIE Proceedings*, vol. 1716, June 1992.

2. Robertson, P. K., and R. G. Campanella, *Guidelines for Geotechnical Design Using the Cone Penetrometer Test and CPT with Pore Pressure Measurement*, Hogentogler & Co., 1989.

3. Wong, T. *CPTINT*, Civil Engineering Department, University of British Columbia, Vancouver, B.C. VGT 1Z4.

4. *DataSculptor and NeuralWorks*, NeuralWare, Inc., Penn Center West, Pittsburgh, PA.

APPENDIX A

SOURCE CODE:

CPNN.CPP
START.CPP
SOIL.C

```
//                              CPNN.CPP
//                Program to Generate Data for Input for CPTINT

//                     by: John Andrews,  12 August 1993
//                        Computer Sciences Corporation
//
//
//   This program creates artificial input data for "CPTINT".  "CPTINT" is
//   a cone data interpretation program distributed by the Civil Engineering
//   Department, University of British Colombia, Vancouver, B.C. Canada
//   phone number: (604) 822-2637.
//
//   Execution of this program creates a file named "nncpt.dat". It contains
//   three space delimited fields, Depth, Qc, and Fs.  A new line separates
//   records. The file "nncpt.dat" can be read directly by "CPTINT".
//
//
//   ------------------------------------------------------------------------



#include <iostream.h>
#include <fstream.h>
#include <math.h>

main()
{

float qc;                                   // cone bearing (bar)
int depth = 0;                              // dummy depth variable

ofstream nncpt("nncpt.dat");                // create file nncpt.dat

 for(float x=0; x<=3; x+=0.03){             // divide y axis into 101 pixels
  qc = pow(10.0,x);                         // compute qc
   for (int n=1; n<=120; n++){              // divide x axis into 120 pixels
    nncpt << depth++ << "  " ;             // writes & increments depth
    nncpt << qc << "  "  ;                 // write qc to file
    nncpt << qc*(.08*n/120) << '\n';        // write fs to file
   }
 }
}
```

```
//                                  START.CPP
//                          by: John Andrews, August 1993
//                          Computer Sciences Corporation
//
//
//   This program reads a NeuralWare *.nnr file having 2 input fields
//   and 12 output fields (each desired and actual).  The input file
//   is entered as a command line argument.  The program creates two files
//   for storing correctly and incorrectly classified records.  It also
//   displays the number of correct and incorrect classifications.
//
//  ------------------------------------------------------------------------


#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <graphics.h>


main(int argc, char *argv[])
{
  float input1, input2, maxvalue;
  float desout[12];
  float actout[12];
  int i, desclass, actclass;
  int numcor = 0, nummis = 0;

  cout << argv[1];

  ifstream nnrfile(argv[1]);              //input file
  if (!nnrfile) {                         //error handling
  cerr << "Can't open the file";
  return(0);
  }

  ofstream corfile("cor.nnr");            //create files for correct
  ofstream misfile("mis.nnr");            //& incorrect records

  while (!nnrfile.eof())  {               //read 2 input fields
    nnrfile >> input1;
    nnrfile >> input2;

    for(i=0; i<12; i++){                  //read 12 desired output fields
        nnrfile >> desout[i];
        if (desout[i] == 1.00)            //determine desired class
          desclass = i;
    }

    nnrfile >> actout[0];                 //read 12 actual output fields
    actclass = 0;
    maxvalue = actout[0];

    for(i=1;  i<12; i++){                 //determine highest actual output
        nnrfile >> actout[i];
        if( actout[i] > maxvalue ){
           actclass = i;
           maxvalue = actout[i];
        }
```

```
    }
                                        //write correctly classified
                                        //records to file
    if (actclass == desclass) {
       numcor++;
       corfile << input1 << "\t" << input2 << "\t" << actclass + 1 <<"\n";
       }
                                        //write incorrectly classified
                                        //records to other file
    else{
       nummis++;
       misfile << input1 << "\t" << input2 << "\t" << actclass + 1 << "\n";
       }


                                    //output results of last
                                    //record (for code testing)

  for(i=0;i<12;i++)
  cout<< desout[i] << "  "  << actout[i] << "\n";
  cout<< desclass << "  "  << actclass << "  " << maxvalue <<"\n\n\n";
  cout << input1 << "  " << input2 << "\n";
  cout << numcor << "  " << nummis << "\n\n"; //output numbers cor & !cor


}
```

```
/*                                SOIL.C                                  */
/*                                                                        */
/*                  by: John Andrews,  September 1993                     */
/*                    Computer Sciences Corporation                       */

/*     This program implements a four layer feedforward neural network   */
/*     for CPT soil classification.                                       */
/*     It takes two parameters, log Qc and Rf (TSF units).                */
/*     It returns soil class integer 1-12.                                */
/*     The network has a 2x15x15x12 architecture.                         */
/*     The output layer is competitive (recall mode).                     */
/*                                                                        */
/*     Mon Sep 20 14:36:49 1993 (cp15.c)                                  */
/*     Recall-Only Run-time for <cp15x15b>                                */
/*     Modification of NeuralWorks Flash code.                            */
/*     Control Strategy is: <backprop>                                    */
/*    ------------------------------------------------------------------  */

  #include <math.h>

  int soil( float cone, float csratio )  /* log cone pressure, sleeve/cone *
  {
    float  Xout[46];  /* work arrays */
    float CmpV;       /* competitive value */
    int   ICmpX;      /* compeititive index */


    /* Read and scale input into network */
    Xout[2] = cone * (0.67342337) + (-1.0328631);
    Xout[3] = csratio * (0.25211143) + (-1.0168915);

    /* Generating code for PE 0 in layer 3 */
    Xout[4]  = (float)(-1.3774354) + (float)(4.3437333) * Xout[2] +
        (float)(0.70192885) * Xout[3];
    Xout[4] = tanh( Xout[4] );

    /* Generating code for PE 1 in layer 3 */
    Xout[5]  = (float)(-0.67682087) + (float)(-6.8595533) * Xout[2] +
        (float)(1.1557226) * Xout[3];
    Xout[5] = tanh( Xout[5] );

    /* Generating code for PE 2 in layer 3 */
    Xout[6]  = (float)(0.96711385) + (float)(-2.6023602) * Xout[2] +
        (float)(0.2549077) * Xout[3];
    Xout[6] = tanh( Xout[6] );

    /* Generating code for PE 3 in layer 3 */
    Xout[7]  = (float)(-1.7575909) + (float)(-8.4847059) * Xout[2] +
        (float)(2.479882) * Xout[3];
    Xout[7] = tanh( Xout[7] );

    /* Generating code for PE 4 in layer 3 */
    Xout[8]  = (float)(5.9350553) + (float)(6.4196491) * Xout[2] +
        (float)(3.760468) * Xout[3];
    Xout[8] = tanh( Xout[8] );

    /* Generating code for PE 5 in layer 3 */
    Xout[9]  = (float)(0.92435199) + (float)(0.45118952) * Xout[2] +
        (float)(-8.390686) * Xout[3];
```

```
Xout[9] = tanh( Xout[9] );

/* Generating code for PE 6 in layer 3 */
Xout[10]  = (float)(-4.9638495) + (float)(9.5670557) * Xout[2] +
    (float)(3.2453654) * Xout[3];
Xout[10] = tanh( Xout[10] );

/* Generating code for PE 7 in layer 3 */
Xout[11]  = (float)(-3.380995) + (float)(2.4127712) * Xout[2] +
    (float)(-8.394412) * Xout[3];
Xout[11] = tanh( Xout[11] );

/* Generating code for PE 8 in layer 3 */
Xout[12]  = (float)(-10.241221) + (float)(-12.071634) * Xout[2] +
    (float)(4.0742674) * Xout[3];
Xout[12] = tanh( Xout[12] );

/* Generating code for PE 9 in layer 3 */
Xout[13]  = (float)(6.7051864) + (float)(-1.3762254) * Xout[2] +
    (float)(7.9877915) * Xout[3];
Xout[13] = tanh( Xout[13] );

/* Generating code for PE 10 in layer 3 */
Xout[14]  = (float)(-2.9832273) + (float)(-0.30514637) * Xout[2] +
    (float)(-6.6823497) * Xout[3];
Xout[14] = tanh( Xout[14] );

/* Generating code for PE 11 in layer 3 */
Xout[15]  = (float)(1.1439002) + (float)(-3.3561134) * Xout[2] +
    (float)(11.728964) * Xout[3];
Xout[15] = tanh( Xout[15] );

/* Generating code for PE 12 in layer 3 */
Xout[16]  = (float)(0.6612637) + (float)(-6.8429189) * Xout[2] +
    (float)(0.44200832) * Xout[3];
Xout[16] = tanh( Xout[16] );

/* Generating code for PE 13 in layer 3 */
Xout[17]  = (float)(3.5646832) + (float)(0.60023791) * Xout[2] +
    (float)(6.4878182) * Xout[3];
Xout[17] = tanh( Xout[17] );

/* Generating code for PE 14 in layer 3 */
Xout[18]  = (float)(0.9410491) + (float)(-4.5457315) * Xout[2] +
    (float)(-0.30045751) * Xout[3];
Xout[18] = tanh( Xout[18] );

/* Generating code for PE 0 in layer 4 */
Xout[19]  = (float)(-0.34009913) + (float)(0.6707626) * Xout[4] +
    (float)(-1.4701272) * Xout[5] + (float)(-0.70717514) * Xout[6] +
    (float)(-1.9541312) * Xout[7] + (float)(3.078223) * Xout[8] +
    (float)(2.21312) * Xout[9] + (float)(-1.4408562) * Xout[10] +
    (float)(-1.6046822) * Xout[11] + (float)(-5.0437288) * Xout[12] +
    (float)(1.2409499) * Xout[13] + (float)(-0.78926444) * Xout[14] +
    (float)(2.8454196) * Xout[15] + (float)(-0.73372024) * Xout[16] +
    (float)(0.87041885) * Xout[17] + (float)(-0.75311923) * Xout[18];
Xout[19] = tanh( Xout[19] );

/* Generating code for PE 1 in layer 4 */
Xout[20]  = (float)(-2.8791444) + (float)(0.48050028) * Xout[4] +
```

```
    (float)(-2.9404998) * Xout[5] + (float)(-0.64095086) * Xout[6] +
    (float)(-2.8091729) * Xout[7] + (float)(0.51963592) * Xout[8] +
    (float)(1.7173361) * Xout[9] + (float)(1.9665118) * Xout[10] +
    (float)(3.4087687) * Xout[11] + (float)(2.4434633) * Xout[12] +
    (float)(-2.1504614) * Xout[13] + (float)(-0.27361864) * Xout[14] +
    (float)(-5.3734245) * Xout[15] + (float)(-0.30357832) * Xout[16] +
    (float)(0.31644058) * Xout[17] + (float)(-0.16554987) * Xout[18];
Xout[20] = tanh( Xout[20] );

/* Generating code for PE 2 in layer 4 */
Xout[21]  = (float)(-1.1246655) + (float)(1.6832063) * Xout[4] +
    (float)(-1.8406967) * Xout[5] + (float)(-1.0049504) * Xout[6] +
    (float)(-0.74303836) * Xout[7] + (float)(1.5295234) * Xout[8] +
    (float)(-1.6239283) * Xout[9] + (float)(2.3027546) * Xout[10] +
    (float)(-2.4416902) * Xout[11] + (float)(-2.1821213) * Xout[12] +
    (float)(-3.2663224) * Xout[13] + (float)(2.1976848) * Xout[14] +
    (float)(1.3920369) * Xout[15] + (float)(-3.0185628) * Xout[16] +
    (float)(-2.4525895) * Xout[17] + (float)(-1.8309394) * Xout[18];
Xout[21] = tanh( Xout[21] );

/* Generating code for PE 3 in layer 4 */
Xout[22]  = (float)(0.96787357) + (float)(-0.88769686) * Xout[4] +
    (float)(1.6487899) * Xout[5] + (float)(0.72958428) * Xout[6] +
    (float)(0.33483446) * Xout[7] + (float)(0.12007745) * Xout[8] +
    (float)(-0.61698985) * Xout[9] + (float)(-1.3958707) * Xout[10] +
    (float)(-1.4732052) * Xout[11] + (float)(-1.1600244) * Xout[12] +
    (float)(0.91249472) * Xout[13] + (float)(-1.8335599) * Xout[14] +
    (float)(1.4040639) * Xout[15] + (float)(2.0465801) * Xout[16] +
    (float)(1.3337294) * Xout[17] + (float)(0.92064232) * Xout[18];
Xout[22] = tanh( Xout[22] );

/* Generating code for PE 4 in layer 4 */
Xout[23]  = (float)(-0.36530879) + (float)(-0.59072089) * Xout[4] +
    (float)(1.2441344) * Xout[5] + (float)(0.21524772) * Xout[6] +
    (float)(1.6768456) * Xout[7] + (float)(-0.50899112) * Xout[8] +
    (float)(0.83034575) * Xout[9] + (float)(-0.94205385) * Xout[10] +
    (float)(-1.5979517) * Xout[11] + (float)(0.24534306) * Xout[12] +
    (float)(0.64400119) * Xout[13] + (float)(-0.18732649) * Xout[14] +
    (float)(0.71088886) * Xout[15] + (float)(0.28635231) * Xout[16] +
    (float)(0.10100901) * Xout[17] + (float)(0.48917824) * Xout[18];
Xout[23] = tanh( Xout[23] );

/* Generating code for PE 5 in layer 4 */
Xout[24]  = (float)(-2.2106702) + (float)(-2.2451408) * Xout[4] +
    (float)(1.1351285) * Xout[5] + (float)(1.4046094) * Xout[6] +
    (float)(1.2378136) * Xout[7] + (float)(0.25965616) * Xout[8] +
    (float)(1.9334576) * Xout[9] + (float)(-1.2937686) * Xout[10] +
    (float)(2.5316298) * Xout[11] + (float)(-2.1072848) * Xout[12] +
    (float)(3.8545687) * Xout[13] + (float)(-2.0066607) * Xout[14] +
    (float)(-3.6767368) * Xout[15] + (float)(1.9547799) * Xout[16] +
    (float)(1.8855346) * Xout[17] + (float)(2.3863194) * Xout[18];
Xout[24] = tanh( Xout[24] );

/* Generating code for PE 6 in layer 4 */
Xout[25]  = (float)(1.4030933) + (float)(0.59960848) * Xout[4] +
    (float)(-1.8639184) * Xout[5] + (float)(-0.3860828) * Xout[6] +
    (float)(-4.0644178) * Xout[7] + (float)(3.9791329) * Xout[8] +
    (float)(6.2324586) * Xout[9] + (float)(3.5871241) * Xout[10] +
    (float)(4.2374911) * Xout[11] + (float)(-3.7616525) * Xout[12] +
    (float)(1.1509675) * Xout[13] + (float)(0.32016569) * Xout[14] +
```

```
    (float)(-0.77581334) * Xout[15] + (float)(-1.4044687) * Xout[16] +
    (float)(-0.62049145) * Xout[17] + (float)(-0.57259399) * Xout[18];
Xout[25] = tanh( Xout[25] );

/* Generating code for PE 7 in layer 4 */
Xout[26]  = (float)(-0.52494174) + (float)(0.5791747) * Xout[4] +
    (float)(-1.5759159) * Xout[5] + (float)(-0.22171608) * Xout[6] +
    (float)(-3.5944927) * Xout[7] + (float)(-0.24836895) * Xout[8] +
    (float)(2.8628707) * Xout[9] + (float)(-1.4358618) * Xout[10] +
    (float)(1.8196349) * Xout[11] + (float)(0.75441694) * Xout[12] +
    (float)(-0.10356036) * Xout[13] + (float)(-0.0011890249) * Xout[14] +
    (float)(-3.3494186) * Xout[15] + (float)(-1.1160185) * Xout[16] +
    (float)(-0.32207805) * Xout[17] + (float)(-0.64801431) * Xout[18];
Xout[26] = tanh( Xout[26] );

/* Generating code for PE 8 in layer 4 */
Xout[27]  = (float)(1.6821843) + (float)(-1.1902723) * Xout[4] +
    (float)(0.99234951) * Xout[5] + (float)(0.76882249) * Xout[6] +
    (float)(0.4855696) * Xout[7] + (float)(1.5828314) * Xout[8] +
    (float)(1.4035158) * Xout[9] + (float)(-4.6815071) * Xout[10] +
    (float)(2.8950787) * Xout[11] + (float)(-5.4909062) * Xout[12] +
    (float)(-2.4956195) * Xout[13] + (float)(2.2027419) * Xout[14] +
    (float)(-0.70475209) * Xout[15] + (float)(2.4308791) * Xout[16] +
    (float)(-2.0018756) * Xout[17] + (float)(1.2343072) * Xout[18];
Xout[27] = tanh( Xout[27] );

/* Generating code for PE 9 in layer 4 */
Xout[28]  = (float)(0.066524662) + (float)(0.2381712) * Xout[4] +
    (float)(0.36230001) * Xout[5] + (float)(-0.32556629) * Xout[6] +
    (float)(0.84901702) * Xout[7] + (float)(-0.0059411367) * Xout[8] +
    (float)(0.22432239) * Xout[9] + (float)(0.20808095) * Xout[10] +
    (float)(-0.76741439) * Xout[11] + (float)(-1.933818) * Xout[12] +
    (float)(-0.28691539) * Xout[13] + (float)(-0.048413601) * Xout[14] +
    (float)(0.45335075) * Xout[15] + (float)(0.39268398) * Xout[16] +
    (float)(-0.35602769) * Xout[17] + (float)(-0.26055622) * Xout[18];
Xout[28] = tanh( Xout[28] );

/* Generating code for PE 10 in layer 4 */
Xout[29]  = (float)(0.20022397) + (float)(-0.35824397) * Xout[4] +
    (float)(-0.10193466) * Xout[5] + (float)(0.44272116) * Xout[6] +
    (float)(-0.25799647) * Xout[7] + (float)(1.3102238) * Xout[8] +
    (float)(-0.054711644) * Xout[9] + (float)(-1.0969514) * Xout[10] +
    (float)(0.17795166) * Xout[11] + (float)(-0.57731533) * Xout[12] +
    (float)(-0.69488567) * Xout[13] + (float)(1.3976555) * Xout[14] +
    (float)(-1.0415714) * Xout[15] + (float)(0.32586724) * Xout[16] +
    (float)(-1.3062199) * Xout[17] + (float)(0.31708151) * Xout[18];
Xout[29] = tanh( Xout[29] );

/* Generating code for PE 11 in layer 4 */
Xout[30]  = (float)(0.91501057) + (float)(0.32980675) * Xout[4] +
    (float)(-0.065754704) * Xout[5] + (float)(0.15424739) * Xout[6] +
    (float)(-0.2029167) * Xout[7] + (float)(-0.37174666) * Xout[8] +
    (float)(0.082694367) * Xout[9] + (float)(0.036902208) * Xout[10] +
    (float)(-0.23626976) * Xout[11] + (float)(3.6132705) * Xout[12] +
    (float)(0.015108704) * Xout[13] + (float)(0.85283083) * Xout[14] +
    (float)(-0.93645924) * Xout[15] + (float)(-0.19907707) * Xout[16] +
    (float)(-0.34797958) * Xout[17] + (float)(-0.17012013) * Xout[18];
Xout[30] = tanh( Xout[30] );

/* Generating code for PE 12 in layer 4 */
```

```
Xout[31]  = (float)(-0.19556384) + (float)(-1.4724393) * Xout[4] +
    (float)(-1.9010726) * Xout[5] + (float)(1.1318318) * Xout[6] +
    (float)(-3.6829324) * Xout[7] + (float)(4.0976496) * Xout[8] +
    (float)(-0.15674546) * Xout[9] + (float)(-0.92516965) * Xout[10] +
    (float)(-0.050786) * Xout[11] + (float)(-1.3713385) * Xout[12] +
    (float)(4.3936687) * Xout[13] + (float)(-2.2437789) * Xout[14] +
    (float)(0.71045846) * Xout[15] + (float)(-0.16340141) * Xout[16] +
    (float)(2.8538706) * Xout[17] + (float)(1.0720434) * Xout[18];
Xout[31] = tanh( Xout[31] );

/* Generating code for PE 13 in layer 4 */
Xout[32]  = (float)(0.79607087) + (float)(1.4678394) * Xout[4] +
    (float)(-3.0694056) * Xout[5] + (float)(-0.62680197) * Xout[6] +
    (float)(-1.2404653) * Xout[7] + (float)(1.1638988) * Xout[8] +
    (float)(-1.7049054) * Xout[9] + (float)(2.3438494) * Xout[10] +
    (float)(1.0563062) * Xout[11] + (float)(-1.2724484) * Xout[12] +
    (float)(-1.7043641) * Xout[13] + (float)(3.1809888) * Xout[14] +
    (float)(1.8637894) * Xout[15] + (float)(-3.2304156) * Xout[16] +
    (float)(-2.3590434) * Xout[17] + (float)(-1.7398988) * Xout[18];
Xout[32] = tanh( Xout[32] );

/* Generating code for PE 14 in layer 4 */
Xout[33]  = (float)(1.5249654) + (float)(-0.061132118) * Xout[4] +
    (float)(1.0983677) * Xout[5] + (float)(0.44974536) * Xout[6] +
    (float)(2.3553016) * Xout[7] + (float)(-1.6995065) * Xout[8] +
    (float)(-2.5550337) * Xout[9] + (float)(2.0845053) * Xout[10] +
    (float)(-0.31993854) * Xout[11] + (float)(-2.2329035) * Xout[12] +
    (float)(-0.79235321) * Xout[13] + (float)(0.25259238) * Xout[14] +
    (float)(1.8125871) * Xout[15] + (float)(-0.27512208) * Xout[16] +
    (float)(-0.17584039) * Xout[17] + (float)(-0.065280482) * Xout[18];
Xout[33] = tanh( Xout[33] );

/* Generating code for PE 0 in layer 5 */
Xout[34]  = (float)(-0.87790149) + (float)(-1.0692201) * Xout[19] +
    (float)(0.03755356) * Xout[20] + (float)(0.096775234) * Xout[21] +
    (float)(-0.049250424) * Xout[22] + (float)(0.10390767) * Xout[23] +
    (float)(-0.096084803) * Xout[24] + (float)(0.014091688) * Xout[25] +
    (float)(0.0051135253) * Xout[26] + (float)(0.06910155) * Xout[27] +
    (float)(-0.041864727) * Xout[28] + (float)(0.064484537) * Xout[29] +
    (float)(-1.089339) * Xout[30] + (float)(-0.084535331) * Xout[31] +
    (float)(-0.016315376) * Xout[32] + (float)(0.021016486) * Xout[33];
Xout[34] = tanh( Xout[34] );
CmpV = Xout[34]; ICmpX = 34; /* start competition */

/* Generating code for PE 1 in layer 5 */
Xout[35]  = (float)(-0.23333962) + (float)(-0.23561044) * Xout[19] +
    (float)(-0.050891768) * Xout[20] + (float)(-0.10643666) * Xout[21] +
    (float)(0.018222565) * Xout[22] + (float)(-0.097888283) * Xout[23] +
    (float)(0.047541238) * Xout[24] + (float)(-0.059924994) * Xout[25] +
    (float)(-0.028140228) * Xout[26] + (float)(-0.12535518) * Xout[27] +
    (float)(0.040437236) * Xout[28] + (float)(-0.0028511814) * Xout[29] +
    (float)(0.92822611) * Xout[30] + (float)(0.21035294) * Xout[31] +
    (float)(0.007625306) * Xout[32] + (float)(-0.068101816) * Xout[33];
Xout[35] = tanh( Xout[35] );
if ( Xout[35] > CmpV ) { CmpV = Xout[35]; ICmpX = 35; } /* track winner */

/* Generating code for PE 2 in layer 5 */
Xout[36]  = (float)(-2.3662107) + (float)(0.93160915) * Xout[19] +
    (float)(-0.11621797) * Xout[20] + (float)(0.024368644) * Xout[21] +
    (float)(-0.14027528) * Xout[22] + (float)(-0.2152873) * Xout[23] +
```

```
    (float)(0.01436625) * Xout[24] + (float)(-1.1697067) * Xout[25] +
    (float)(0.037573472) * Xout[26] + (float)(0.011277422) * Xout[27] +
    (float)(0.094070829) * Xout[28] + (float)(0.05417366) * Xout[29] +
    (float)(-0.16597992) * Xout[30] + (float)(1.2443105) * Xout[31] +
    (float)(-0.14807294) * Xout[32] + (float)(0.020404585) * Xout[33];
Xout[36] = tanh( Xout[36] );
if ( Xout[36] > CmpV ) { CmpV = Xout[36]; ICmpX = 36; } /* track winner */

/* Generating code for PE 3 in layer 5 */
Xout[37]  = (float)(-1.1655524) + (float)(0.14780924) * Xout[19] +
    (float)(0.20306671) * Xout[20] + (float)(-0.051036075) * Xout[21] +
    (float)(-0.59072953) * Xout[22] + (float)(0.22242334) * Xout[23] +
    (float)(0.1092864) * Xout[24] + (float)(0.98674947) * Xout[25] +
    (float)(-1.0761287) * Xout[26] + (float)(0.11027345) * Xout[27] +
    (float)(0.086110197) * Xout[28] + (float)(-0.13758875) * Xout[29] +
    (float)(0.29757303) * Xout[30] + (float)(-0.018368572) * Xout[31] +
    (float)(-0.53483301) * Xout[32] + (float)(-0.22306475) * Xout[33];
Xout[37] = tanh( Xout[37] );
if ( Xout[37] > CmpV ) { CmpV = Xout[37]; ICmpX = 37; } /* track winner */

/* Generating code for PE 4 in layer 5 */
Xout[38]  = (float)(-2.9970107) + (float)(1.1596954) * Xout[19] +
    (float)(-1.1503054) * Xout[20] + (float)(0.005177083) * Xout[21] +
    (float)(-0.096239172) * Xout[22] + (float)(-0.024488389) * Xout[23] +
    (float)(-0.051597442) * Xout[24] + (float)(0.28520513) * Xout[25] +
    (float)(1.0015672) * Xout[26] + (float)(0.81431431) * Xout[27] +
    (float)(-0.1415341) * Xout[28] + (float)(0.026307182) * Xout[29] +
    (float)(-0.053032927) * Xout[30] + (float)(-0.011248631) * Xout[31] +
    (float)(-0.10334851) * Xout[32] + (float)(0.28688434) * Xout[33];
Xout[38] = tanh( Xout[38] );
if ( Xout[38] > CmpV ) { CmpV = Xout[38]; ICmpX = 38; } /* track winner */

/* Generating code for PE 5 in layer 5 */
Xout[39]  = (float)(-1.6181245) + (float)(0.03928183) * Xout[19] +
    (float)(0.9619233) * Xout[20] + (float)(0.0081323469) * Xout[21] +
    (float)(0.34299639) * Xout[22] + (float)(-0.15945068) * Xout[23] +
    (float)(-0.1256876) * Xout[24] + (float)(0.59323692) * Xout[25] +
    (float)(0.030090949) * Xout[26] + (float)(0.044754967) * Xout[27] +
    (float)(-0.067690797) * Xout[28] + (float)(0.001090379) * Xout[29] +
    (float)(0.016745619) * Xout[30] + (float)(0.56472063) * Xout[31] +
    (float)(-0.83290011) * Xout[32] + (float)(0.6618259) * Xout[33];
Xout[39] = tanh( Xout[39] );
if ( Xout[39] > CmpV ) { CmpV = Xout[39]; ICmpX = 39; } /* track winner */

/* Generating code for PE 6 in layer 5 */
Xout[40]  = (float)(-1.0943762) + (float)(-0.035113763) * Xout[19] +
    (float)(-0.24052712) * Xout[20] + (float)(-1.1928893) * Xout[21] +
    (float)(-0.30317307) * Xout[22] + (float)(-0.15959917) * Xout[23] +
    (float)(-0.07071659) * Xout[24] + (float)(0.0035665662) * Xout[25] +
    (float)(-0.014620778) * Xout[26] + (float)(-0.01583972) * Xout[27] +
    (float)(-0.052799445) * Xout[28] + (float)(0.0028659943) * Xout[29] +
    (float)(-0.098308317) * Xout[30] + (float)(0.010499664) * Xout[31] +
    (float)(0.93895841) * Xout[32] + (float)(0.01171761) * Xout[33];
Xout[40] = tanh( Xout[40] );
if ( Xout[40] > CmpV ) { CmpV = Xout[40]; ICmpX = 40; } /* track winner */

/* Generating code for PE 7 in layer 5 */
Xout[41]  = (float)(-1.528774) + (float)(0.044951025) * Xout[19] +
    (float)(-0.24355549) * Xout[20] + (float)(1.1148211) * Xout[21] +
    (float)(0.20645723) * Xout[22] + (float)(-0.12182666) * Xout[23] +
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1994	Final

4. TITLE AND SUBTITLE

DEVELOPMENT OF AN ARTIFICIAL NEURAL NETWORK FOR REAL-TIME CLASSIFICATION OF CONE PENETROMETER STRAIN GAUGE DATA

5. FUNDING NUMBERS

PE: 0603716N
WU: DN300197

6. AUTHOR(S)

John M. Andrews and Stephen H. Lieberman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Command, Control and Ocean Surveillance Center (NCCOSC)
RDT&E Division
San Diego, CA 92152–5001

8. PERFORMING ORGANIZATION REPORT NUMBER

TR 1682

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Arlington, VA 22217

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Technology detailed in this document is covered by a U.S. Patent Application assigned to the U.S. Government. Parties interested in licensing this technology may direct inquiries to:

Harvey Fendelman
Legal Counsel for Patents
Code 0012
NCCOSC, RDT&E Division
San Diego, CA 92152–5765
(619) 553–3001

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document describes the development of an artificial neural-network-based algorithm for classifying soil behavior type from cone penetrometer strain gauge data. The network input consists of the two standard cone penetration test parameters: the logarithm of cone pressure and the percentage of sleeve friction to cone pressure (friction ratio). Network output is a one-of-n coding of 12 soil classifications. Three- and four-layer backpropagation networks are trained to associate over 11,000 data points with the appropriate soil type. The best recall performance is obtained from a four-layer, 2x15x15x12 network with a tested accuracy rate of 98.2%. All classification errors occur at the decision boundaries between class regions. The network was incorporated into the data collection software of the prototype SCAPS vehicle in October 1993. The C source code is included as appendix A.

14. SUBJECT TERMS

Site Characterization and Analysis Penetrometer System (SCAPS)
soil behavior type
cone penetration test
C source code

15. NUMBER OF PAGES

30

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a. NAME OF RESPONSIBLE INDIVIDUAL	21b. TELEPHONE *(include Area Code)*	21c. OFFICE SYMBOL
Steven H. Lieberman	(619) 553–2778	Code 521

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0271	Archive/Stock	(6)
Code 0274	Library	(2)
Code 521	J. M. Andrews	(100)

Defense Technical Information Center
Alexandria, VA 22304–6145 (4)